

Apple Assembly Line

Volume 2 -- Issue 1

October, 1981

In This Issue...

Sifting Primes Faster and Faster	2
6809 Cross Assembler	12
Extending the Apple Monitor	14
Errata	18
DOS 3.3 Disassembly: \$B052-B0B5 and \$B35F-B7FF	18

Source Code for S-C Assembler II Version 4.0

At long last, I have decided to start selling the source code for my assembler. So many of you have asked for it! I am sure you understand my reluctance; after all, with a wife and five kids to support, and most of our income coming from this one product....

If I have your registration card for Version 4.0 on file, or some other proof-of-purchase, I will send you a disk with all of the commented source code on it. You can study it, assemble it, modify it, et cetera; just don't start selling it! With your check for \$95, you will need to include the following signed declaration:

"I am purchasing the source code of S-C Assembler II Version 4.0 with the understanding that it is proprietary information belonging to S-C SOFTWARE. The disk, and any copies or listings I may make of it, are only for my own personal use."

Another Way Out of the Assembler

James Church, from Trumbull, CT, writes that he has found a way to get from the assembler into Applesoft, without wiping out an Applesoft program.

The normal way to leave is by typing FP, and then PR#0. This of course clears any Applesoft program from memory. But by typing \$AAB6:40, \$E003G, and PR#0 you can enter Applesoft softly.

Sifting Primes Faster and Faster

Benchmark programs are sometimes useful for selecting between various processors. Quite a few articles have been published which compare and rank the various Z-80, 8080, 6800, and 6502 systems based on the speed with which they execute a given BASIC program. Some of us cannot resist the impulse to show them up by recoding the benchmark in our favorite language on our favorite processor, using our favorite secret tricks for trimming microseconds.

"A High-Level Language Benchmark" (by Jim Gilbreath, BYTE, September, 1981, pages 180-198) is just such an article. Jim compared execution time in Assembly, Forth, Basic, Fortran, COBOL, PL/I, C, and other languages; he used all sorts of computers, including the above four, the Motorola 68000, the DEC PDP 11/70, and more. He used a short program which finds the 1899 primes between 3 and 16384 by means of a sifting algorithm (Sieve of Eratosthenes).

His article includes table after table of comparisons. Some of the key items of interest to me were:

Language and Machine	Seconds
Assembly Language 68000 (8 MHz)	1.12
Assembly Language Z80	6.80
Digital Research PL/I (Z80)	14.0
Microsoft BASIC Compiler (Z80)	18.6
FORTH 6502	265.
Apple UCSD Pascal	516.
Apple Integer BASIC	2320.
Applesoft BASIC	2806.
Microsoft COBOL Version 2.2 (Z80)	5115.

There is a HUGE error in the data above; I don't know if it is the only one or not. The time I measured for the Apple Integer BASIC version was only 188 seconds, not 2320 seconds! How could he be so far off? His data is obviously wrong, because Integer BASIC in his data is too close to the same speed as Applesoft.

I also don't know why they neglected to show what the 6502 could do with an assembly language version. Or maybe I do....were they ashamed?

William Robert Savoie, an Apple owner from Tennessee, sent me a copy of the article along with his program. He "hand-compiled" the BASIC version of the benchmark program, with no special tricks at all. His program runs in only 1.39 seconds! That is almost as fast as the 8 MHz Motorola 68000 system! The letter that accompanied his program challenged anyone to try to speed up his program.

How could I pass up a challenge like that? I wrote my own version of the program, and cut the time to .93 seconds! Then I made one small change to the algorithm, and produced exactly the same results in only .74 seconds!

Looking back at Jim Gilbreath's article, he concludes that efficient, powerful high-level languages are THE way to go. He eschews the use of assembly language for any except the most drastic requirements, because he could not see a clear speed advantage. He points out the moral that a better algorithm is superior to a faster CPU. (Note that his algorithm is by no means the fastest one, by the way.)

Here is Gilbreath's algorithm, in Integer BASIC:

```
>LIST
10 S=8190: DIM F(8191):N=0
20 FOR I=0 TO S:F(I)=1: NEXT I
30 FOR I=0 TO S: IF F(I)=0 THEN 80
40 P=I+I+3:K=I+P
50 IF K>S THEN 70
60 F(K)=0:K=K+P: GOTO 50
70 N=N+1: REM PRINT P;" ";
80 NEXT I
90 PRINT : PRINT N;" PRIMES": END
```

NEW UTILITIES FOR S-C ASSEMBLER

GLOBAL SEARCH & REPLACE

- * REPLACES LABEL NAMES QUICKLY AND EASILY
- * SEARCH ALL OR PART OF SOURCE CODE
- * OPTIONAL PROMPTING FOR USER VERIFICATION
- * PROGRAM DISKETTE + DOCUMENTATION: \$ 20.00

CROSS REFERENCE TABLE

- * A COMPLETE CROSS REFERENCE OF GLOBAL LABELS BY LINE #
- * TABLE GENERATED IN ALPHABETICAL ORDER
- * LEADING LABEL LINE NUMBERS HIGHLIGHTED
- * SEE EXAMPLE OUTPUT IN AD OF APRIL 'APPLE ASSEMBLY LINE'
- * PROGRAM DISKETTE AND DOCUMENTATION: \$ 20.00

THE ABOVE MACHINE LANGUAGE UTILITIES ARE FOR USE WITH THE
S-C ASSEMBLER VERSION 4.0

R A K - W A R E
41 Ralph Road
West Orange, NJ 07052

The REM tagged onto the end of line 70, if changed to a real PRINT statement, will print the list of prime numbers as they are generated. Of course printing them was not included in any of the time measurements. According to my timing, printing adds 12 seconds to the program.

I modified the algorithm to take advantage of some more prior knowledge about sifting: There is no need to go through the loop in lines 50 and 60 if P is greater than 127 (the largest prime no bigger than the square root of 16384). This means changing line 40 to read:

```
40 P=I+I+3 : IF P>130 THEN 70 : K=I+P
```

This change cut the time for the program from 188 seconds to 156 seconds. My assembly language version of the original algorithm ran in .93 seconds, or 202 times faster; the better algorithm ran in .74 seconds, or almost 211 times faster.

William Savoie has done a magnificent job in hand-compiling the first program. He ran the program 100 times in a loop, so that he could get an accurate time using his Timex watch. Here is the listing of his program.

write now

Southwestern Data Systems, an industry pioneer in innovative software for the Apple II, is always looking for authors. There are no limitations on the size or type of software you can submit — utilities, communication, business, education, or games — the only requirement is that it must meet the quality standards which typify all SDS products. When you join the SDS team, you get the benefits of a professional support staff experienced in providing all you need to get your program to market. Here are some of the ways we help you:

- TECHNICAL PROGRAMMING ASSISTANCE
- UNIQUE COPY PROTECTION W/LIMITED BACKUPS
- SUCCESSFUL MARKETING STRATEGIES
- ASSISTANCE IN WRITING THE MANUAL
- PROFESSIONAL PRODUCT ARTWORK
- QUALITY ADVERTISING
- SUPERIOR PACKAGING
- NATIONAL DISTRIBUTION
- HIGHEST ROYALTIES PAID MONTHLY
- CUSTOMER SERVICE SUPPORT

This is the opportunity you have been waiting for, a chance to market your program with the finest publisher in the software industry. Let Southwestern Data Systems' reputation and proven track record for success go to work for you. If you think you have what we want — a unique and distinctive software package — please call or write us today!

SDS southwestern data systems

P.O. BOX 582 SANTEE, CA 92071 (714) 562-3670

```

1000 *
1010 * SIEVE PROGRAM:
1020 * CALCULATES FIRST 1899 PRIMES IN 1.39 SECONDS!
1030 *
1040 * INSPIRED BY JIM GILBREATH, BYTE, 9/81
1050 *
1060 * WRITTEN BY WILLIAM ROBERT SAVOIE
1070 * 4405 DELASHMITT RD, APT 15
1080 * HIXSON, TENN 37343
1090 *
3500- 1100 BUFF .EQ $3500 START OF BUFFER (#BUFF=0)
1FFD- 1110 SIZE .EQ 8189 SIZE OF FLAG ARRAY
1120 *
1130 * PAGE-ZERO VARIABLES
1140 *
0006- 1150 INDEX .EO $06 PAGE ZERO INDEX (LOCATION FOR I)
0008- 1160 PRIME .EO $08 PRIME LOCATION
0019- 1170 KVAR .EO $19 K VARIABLE
001B- 1180 CVAR .EO $1B COUNT OF PRIME
001D- 1190 ARRAY .EO $1D ARRAY POINTER
001F- 1200 SAVE .EO $1F COUNT LOOP
1210 *
1220 * ROM ROUTINES
1230 *
FC58- 1240 HOME .EO SFC58 CLEAR VIDEO
FD8E- 1250 CR .EO SFD8E CARRIAGE RETURN
FD9E- 1260 LINE .EO SFD9E PRINT "-"
F940- 1270 PRINTN .EO SF940 PRINT 2 BYTE NUMBER IN HEX
FBE2- 1280 BELL .EQ $FBE2 SOUND BELL WHEN DONE
1290 *
1300 * RUN PROGRAM 100 TIMES FOR ACCURATE TIME MEASUREMENTS!
1310 *
0800- 20 58 FC 1320 START JSR HOME CLEAR SCREEN
0803- 20 8E FD 1330 JSR CR CARRIAGE RETURN
0806- A9 64 1340 LDA #100 LOOP 100 TIMES
0808- 85 1F 1350 STA SAVE SET COUNTER
080A- 20 18 08 1360 .01 JSR GO RUN PRIME
080D- C6 1F 1370 DEC SAVE DECREASE SAVE
080F- D0 F9 1380 BNE .01 LOOP
0811- 20 BA 08 1390 JSR PRINT PRINT COUNT
0814- 20 E2 FB 1400 JSR BELL READ WATCH!
0817- 60 1410 RTS

```

APPLE 8-BIT 8-CHANNEL A/D SYSTEM

- 8-BIT RESOLUTION
- ON BOARD MEMORY-
(Just peek at data)
- FAST CONVERSION -
(.078 ms per channel).
- ELIMINATES NEED TO WAIT FOR A/D CON-
VERSION
- A/D PROCESS TOTALLY TRANSPARENT
TO APPLE.
- FULL SCALE INPUTS CAN EASILY BE
CHANGED BY USER.

APPLIED ENGINEERING'S A/D board is a breakthrough product for all APPLE owners giving real world data at a really affordable price. Diverse applications include monitoring of:

.....TEMPERATURE.....HUMIDITY.....WIND SPEED.....WIND DIRECTION.....
LIGHT INTENSITY.....PRESSURE.....RPM.....SOIL MOISTURE.....
AND MANY MORE.....

CONTRIBUTED PROGRAMS ARE DISTRIBUTED FREE TO ALL A/D OWNERS IN OUR NEWSLETTER.

See your dealer or contact -

APPLIED ENGINEERING
 P.O. BOX 470301
 DALLAS, TEXAS 75247

\$129

MASTER CHARGE & VISA WELCOME



(214) 492-2027



7:00 AM - 11:00 PM 7 DAYS A WEEK

APPLE PERIPHERALS ARE OUR ONLY BUSINESS

```

1420 *
1430 * RESET VARIABLES
1440 *
0818- A0 00 1450 GO LDY #00 CLEAR INDEX
081A- 84 1B 1460 STY CVAR CLEAR COUNT VARIABLE
081C- 84 1C 1470 STY CVAR+1 HI BYTE TOO
081E- 84 06 1480 STY INDEX CLEAR INDEX
0820- 84 07 1490 STY INDEX+1 HI BYTE TOO
0822- 84 1D 1500 STY ARRAY LOW BYTE OF ARRAY
0824- A9 35 1510 LDA /BUFF GET BUFFER LOCATION
0826- 85 1E 1520 STA ARRAY+1 SET ARRAY POINTER
0828- A9 01 1530 LDA #S01 LOAD WITH ONE
082A- A2 1F 1540 LDX /SIZE LOAD STOP BYTE
082C- E8 1550 INX MAKE PAGE LARGER
1560 *
1570 * SET EACH ELEMENT IN ARRAY TO ONE
1580 *
082D- 91 1D 1590 SET STA (ARRAY),Y SET MEMORY
082F- 88 1600 DEY NEXT LOCATION
0830- D0 FB 1610 BNE SET GO 256 TIMES
0832- E6 1E 1620 INC ARRAY+1 MOVE ARRAY INDEX
0834- CA 1630 DEX TEST END
0835- D0 F6 1640 BNE SET GO TELL END
1650 *
1660 * SET ARRAY INDEX AT START OF BUFFER
0837- A9 00 1670 LDA #BUFF SET BUFFER LOCATION
0839- 85 1D 1680 STA ARRAY IN ARRAY POINTER LOW
083B- A9 35 1690 LDA /BUFF SET BUFFER LOCATION
083D- 85 1E 1700 STA ARRAY+1 IN ARRAY POINTER
083F- 4C 48 08 1710 JMP FORIN ENTER SIEVE ALGORITHM
1720 *
1730 * SCAN ENTIRE ARRAY AND PROBAGATE LAST PRIME
0842- E6 06 1740 FORNKT INC INDEX INCREASE LOW BYTE
0844- D0 02 1750 BNE FORIN GO IF < 256
0846- E6 07 1760 INC INDEX+1 INCREASE HI BYTE
0848- A5 06 1770 FORIN LDA INDEX GET INDEX TO ARRAY
084A- 18 1780 CLC READY ADD
084B- 85 1D 1790 STA ARRAY SAVE LOW BYTE
084D- A5 07 1800 LDA INDEX+1 GET HI BYTE
084F- 69 35 1810 ADC /BUFF ADD BUFFER LOCATION
0851- 85 1E 1820 STA ARRAY+1 SET POINTER
0853- A0 00 1830 LDY #00 CLEAR Y REGISTER
0855- B1 1D 1840 LDA (ARRAY),Y GET ARRAY VALUE
0857- F0 E9 1850 BEO FORNKT GO IF FLAG=0 SINCE NOT PRIME
1860 *
1870 * CALCULATE NEXT PRIME NUMBER WITH P=I+I+3
0859- A5 06 1870 LDA INDEX MAKE P=I+3
085B- 69 03 1880 ADC #03 ADD THREE
085D- 85 08 1890 STA PRIME
085F- A5 07 1900 LDA INDEX+1
0861- 69 00 1910 ADC #00 ADD CARRY
0863- 85 09 1920 STA PRIME+1
1930 *
1940 * NOW P=I+3
0865- A5 08 1940 LDA PRIME
0867- 65 06 1950 ADC INDEX MAKE P=P+I
0869- 85 08 1960 STA PRIME
086B- A5 09 1970 LDA PRIME+1
086D- 65 07 1980 ADC INDEX+1 ADD HI BYTE
086F- 85 09 1990 STA PRIME+1 SAVE P
2000 *
2010 * NOW CALCULATE K=I+PRIME (CLEAR BEYOND PRIME)
0871- A5 06 2020 LDA INDEX ADD I TO P
0873- 65 08 2030 ADC PRIME
0875- 85 19 2040 STA KVAR SAVE IN K
0877- A5 07 2050 LDA INDEX+1
0879- 65 09 2060 ADC PRIME+1 ADD HI BYTE TOO
087B- 85 1A 2070 STA KVAR+1 SAVE K VALUE
2080 *
2090 * SEE IF K > SIZE AND MODIFY ARRAY IF NOT
087D- A5 19 2100 .02 LDA KVAR GET K VAR
087F- 38 2110 SEC SET CARRY FOR SUB
0880- E9 FD 2120 SBC #SIZE SUBTRACT SIZE
0882- A5 1A 2130 LDA KVAR+1 GET HI BYTE
0884- E9 1F 2140 SBC /SIZE SUBTRACT TOO
0886- B0 1E 2150 BCS .03 GO IF K < SIZE
2160 *
2170 * ASSIGN ARRAY(K)=0 SINCE PRIME CAN BE ADDED TO MAKE NUMBER
2180 * THEREFORE THIS CANNOT BE PRIME! (PROBAGATE THROUGH ARRAY)
0888- A5 19 2180 LDA KVAR GET INDEX TO ARRAY
088A- 85 1D 2190 STA ARRAY SAVE LOW BYTE
088C- A5 1A 2200 LDA KVAR+1 GET HI BYTE
088E- 69 35 2210 ADC /BUFF ADD BUFFER OFFSET
0890- 85 1E 2220 STA ARRAY+1 SAVE ARRAY INDEX

```

```

0892- A9 00      2230      LDA #00          CLEAR A
0894- A8          2240      TAY              AND Y REGISTER
0895- 91 1D      2250      STA (ARRAY),Y    CLEAR ARRAY LOCATION
                                * CREATE NEW K FROM K=K+PRIME (MOVE THROUGH ARRAY)
0897- A5 19      2270      LDA KVAR          GET K LOW
0899- 65 08      2280      ADC PRIME         ADD PRIME
089B- 85 19      2290      STA KVAR          SAVE K
089D- A5 1A      2300      LDA KVAR+1        NOW ADD HI BYTES
089F- 65 09      2310      ADC PRIME+1
08A1- 85 1A      2320      STA KVAR+1
08A3- 4C 7D 08   2330      JMP .02          LOOP TELL ARRAY DONE
                                * NOW COUNT PRIMES FOUND (C=C+1)
                                .03
08A6- 20 C2 08   2360      * --NOTE-- DELETE NEXT LINE TO TIME PROGRAM (JSR PRINTP)
08A9- E6 1B      2370      JSR PRINTP        PRINT PRIME
08AB- D0 02      2380      INC CVAR          ADD ONE
08AD- E6 1C      2390      BNE .04          GO IF NO OVERFLOW
08AF- A5 06      2400      INC CVAR+1        HI BYTE COUNTER
                                .04
08B1- E9 FD      2410      LDA INDEX        GET INDEX
08B3- A5 07      2420      * TEST TO SEE IF WE HAVE INDEXED THROUGH ENTIRE ARRAY
08B5- E9 1F      2430      SBC #SIZE        SUBTRACT SIZE
08B7- 90 89      2440      LDA INDEX+1      GET HI BYTE TOO
08B9- 60          2450      SBC /SIZE       SUBTRACT HI BYTE
                                2460      BCC FORNXT   CONTINUE?
                                2470      RTS
                                *
                                2480      * PRINT THE NUMBER OF PRIMES FOUND
                                2490      *
08BA- A4 1C      2500      PRINT LDY CVAR+1  GET HI BYTE OF COUNT
08BC- A6 1B      2510      LDX CVAR
08BE- 20 40 F9   2520      JSR PRINTN      PRINT PRIMES FOUND
08C1- 60          2530      RTS            JOB DONE, RETURN
                                *
                                2540      * PRINT THE PRIME NUMBER (OPTIONAL)
                                2550      *
08C2- A4 09      2560      PRINTP LDY PRIME+1 HI BYTE
08C4- A6 08      2570      LDX PRIME
08C6- 20 40 F9   2580      JSR PRINTN
08C9- 20 9E FD   2590      JSR LINE        VIDEO "--" OUT
08CC- 38          2600      SEC
08CD- 60          2610      RTS

```

APPLE MUSIC SYNTHESIZER BREAKTHROUGH

- COMPLETE 16 VOICE MUSIC SYNTHESIZER ON ONE CARD. JUST PLUG IT INTO YOUR APPLE. CONNECT THE AUDIO CABLE (SUPPLIED) TO YOUR STEREO AND BOOT THE SUPPLIED DISK AND YOU'RE READY TO ENTER AND PLAY SONGS.
- IT'S EASY TO PROGRAM MUSIC WITH OUR "COMPOSE" SOFTWARE. YOU'LL START RIGHT AWAY AT INPUTTING YOUR FAVORITE SONGS. OUR MANUAL SHOWS YOU HOW, STEP BY STEP. THE HI-RES SCREEN SHOWS WHAT YOU'VE ENTERED IN STANDARD SHEET MUSIC FORMAT.
- WE GIVE YOU LOTS OF SOFTWARE. IN ADDITION TO "COMPOSE" AND PLAY PROGRAMS, THE DISK IS FULL OF SONGS READY TO RUN.
- FOUR WHITE NOISE GENERATORS (GREAT FOR SOUND EFFECTS).
- PLAYS MUSIC IN TRUE STEREO AS WELL AS TRUE DISCREET QUADRAPHONIC.
- ENVELOPE CONTROL (VOLUME)
- WILL PLAY SONGS WRITTEN FOR ALF SYNTHESIZER (ALF SOFTWARE WILL NOT TAKE ADVANTAGE OF ALL THE FEATURES OF THIS BOARD, THEIR SOFTWARE SOUNDS THE SAME ON OUR SYNTHESIZER).
- AUTOMATIC SHUTOFF ON POWER-UP, OR IF RESET IS PUSHED.
- MANY, MANY MORE FEATURES.

ALL ORDERS SHIPPED SAME DAY
SEND \$150.00 CHECK OR MONEY ORDER
(TEXAS RESIDENTS ADD 5% SALES TAX)

APPLIED ENGINEERING
P.O. BOX 470301
DALLAS, TEXAS 75247

MASTER CHARGE & VISA WELCOME



(214) 492-2027



7:00 AM - 11:00 PM 7 DAYS A WEEK
APPLE PERIPHERALS ARE OUR ONLY BUSINESS

Here is a listing of my fastest version.

```

1000 *
1010 * SIEVE PROGRAM:
1020 * CALCULATES FIRST 1899 PRIMES IN .74 SECONDS!
1030
1040 * INSPIRED BY JIM GILBREATH
1050 * (SEE BYTE MAGAZINE, 9/81, PAGES 180-198.)
1060 * AND BY WILLIAM ROBERT SAVOIE
1070 * 4405 DELASHMITT RD, APT 15
1080 * HIXSON, TENN 37343
1090 *
3500- 1100 ARRAY .EQ $3500 FLAG BYTE ARRAY
2000- 1110 SIZE .EQ 8192 SIZE OF FLAG ARRAY
1120 *
1130 * PAGE-ZERO VARIABLES
1140 *
0006- 1150 A.PNTR .EQ $06,07 POINTER TO FLAG ARRAY FOR OUTER LOOP
0008- 1160 B.PNTR .EQ $08,09 POINTER TO FLAG ARRAY FOR INNER LOOP
001B- 1170 PRIME .EQ $1B,1C LATEST PRIME NUMBER
001D- 1180 COUNT .EQ $1D,1E # OF PRIMES SO FAR
001F- 1190 TIMES .EQ $1F COUNT LOOP
1200 *
1210 * APPLE ROM ROUTINES USED
1220 *
F940- 1230 PRINTN .EQ $F940 PRINT 2 BYTE NUMBER FROM MONITOR
FC58- 1240 HOME .EQ $FC58 CLEAR VIDEO
FD8E- 1250 CR .EQ $FD8E CARRIAGE RETURN
FD9E- 1260 LINE .EQ $FD9E PRINT "-"
FBE2- 1270 BELL .EQ $FBE2 SOUND BELL WHEN DONE
1280 *
1290 * RUN PROGRAM 100 TIMES FOR ACCURATE TIME MEASUREMENTS!
1300 *
0800- 20 58 FC 1310 START JSR HOME CLEAR SCREEN
0803- A9 64 1320 LDA #100 LOOP 100 TIMES
0805- 85 1F 1330 STA TIMES SET COUNTER
0807- 20 21 08 1340 .1 JSR GENERATE.PRIMES
080A- AD 00 04 1350 LDA $400 TOGGLE SCREEN FOR VISIBLE INDICATION
080D- 49 80 1360 EOR $S80 THAT PROGRAM IS STILL RUNNING
080F- 8D 00 04 1370 STA $400
0812- C6 1F 1380 DEC TIMES
0814- D0 F1 1390 BNE 1 LOOP
0814- D0 F2 FB 1400 JSR BELL READ WATCH!
0818- A4 1E 1410 LDY COUNT+1 GET HI BYTE OF COUNT
081B- A6 1D 1420 LDX COUNT
081D- 20 40 F9 1430 JSR PRINTN PRINT PRIMES FOUND
0820- 60 1440 RTS
1450 *
1460 * GENERATE THE PRIMES
1470 *
1480 GENERATE.PRIMES
0821- A0 00 1490 LDY #0 CLEAR INDEX
0823- 84 1D 1500 STY COUNT CLEAR COUNT VARIABLE
0825- 84 1E 1510 STY COUNT+1
0827- 84 06 1520 STY A.PNTR SET UP POINTER FOR OUTER LOOP
0829- A9 35 1530 LDA /ARRAY
082B- 85 07 1540 STA A.PNTR+1
082D- A9 01 1550 LDA #1 LOAD WITH ONE
082F- A2 20 1560 LDX /SIZE NUMBER OF PAGES TO STORE IN
1570 *
1580 * SET EACH ELEMENT IN ARRAY TO ONE
1590 *
0831- 91 06 1600 .1 STA (A.PNTR),Y SET FLAG TO 1
0833- C8 1610 INY NEXT LOCATION
0834- D0 FB 1620 BNE .1 GO 256 TIMES
0836- E6 07 1630 INC A.PNTR+1 POINT AT NEXT PAGE
0838- CA 1640 DEX NEXT PAGE
0839- D0 F6 1650 BNE .1 MORE PAGES
1660 *
1670 * SCAN ENTIRE ARRAY, LOOKING FOR A PRIME
1680 *
083B- A9 35 1690 LDA /ARRAY SET A.PNTR TO BEGINNING AGAIN
083D- 85 07 1700 STA A.PNTR+1
083F- A0 00 1710 .2 LDY #0 CLEAR INDEX
0841- B1 06 1720 LDA (A.PNTR),Y LOOK AT NEXT FLAG
0843- F0 44 1730 BEQ .6 NOT PRIME, ADVANCE POINTER

```



```

1740 *
1750 * CALCULATE CURRENT INDEX INTO FLAG ARRAY
1760 *
0845- 38      1770      SEC
0846- A5 07    1780      LDA A.PNTR+1
0848- E9 35    1790      SEC /ARRAY
084A- AA      1800      TAX      SAVE HI-BYTE OF INDEX
084B- A5 06    1810      LDA A.PNTR      LO-BYTE OF INDEX
1820 *
1830 * CALCULATE NEXT PRIME NUMBER WITH P=I+I+3
1840 *
084D- 0A      1850      ASL      DOUBLE THE INDEX
084E- A8      1860      TAY
084F- 8A      1870      TXA      HI-BYTE OF INDEX
0850- 2A      1880      ROL
0851- AA      1890      TAX
0852- 98      1900      TYA      NOW ADD 3
0853- 69 03    1910      ADC #3
0855- 85 1B    1920      STA PRIME
0857- 90 01    1930      BCC .3
0859- E8      1940      INX
085A- 86 1C    1950      STX PRIME+1
1960 *
1970 * FOLLOWING 4 LINES CHANGE ALGORITHM SLIGHTLY
1980 * TO SPEED IT UP FROM .93 TO .74 SECONDS
1990 *
085C- 8A      2000      TXA      TEST HIGH BYTE
085D- D0 24    2010      BNE .5      PRIME > SQRT(16384)
085F- C0 7F    2020      CPY #127
0861- B0 20    2030      BCS .5      PRIME > SQRT(16384)
2040 *
2050 * NOW CLEAR EVERY P-TH ENTRY AFTER P
2060 *
0863- A0 00    2070      LDY #0
0865- A5 06    2080      LDA A.PNTR      USE CURRENT OUTER POINTER FOR
0867- 85 08    2090      STA B.PNTR      INNER POINTER
0869- A5 07    2100      LDA A.PNTR+1
086B- 85 09    2110      STA B.PNTR+1
086D- 18      2120      CLC      BUMP ARRAY POINTER BY P
086E- A5 08    2130      LDA B.PNTR      BUMP TO NEXT SLOT
0870- 65 1B    2140      ADC PRIME
0872- 85 08    2150      STA B.PNTR
0874- A5 09    2160      LDA B.PNTR+1
0876- 65 1C    2170      ADC PRIME+1
0878- 85 09    2180      STA B.PNTR+1
087A- C9 55    2190      CMP /ARRAY+SIZE      SEE IF BEYOND END OF ARRAY
087C- B0 05    2200      BCS .5      YES, FINISHED CLEARING
087E- 98      2210      TYA      NO, CLEAR ENTRY IN ARRAY
087F- 91 08    2220      STA (B.PNTR),Y
0881- F0 EB    2230      BEQ .4      ...ALWAYS
2240 *
2250 * NOW COUNT PRIMES FOUND (C=C+1)
2260 *
2270      .5
0883- E6 1D    2280      JSR PRINTP      PRINT PRIME
0885- D0 02    2290      INC COUNT
0887- E6 1E    2300      BNE .6
2310      INC COUNT+1
2320 *
2330 * ADVANCE OUTER POINTER AND TEST IF FINISHED
2340 *
0889- E6 06    2350      .6      INC A.PNTR
088B- D0 02    2360      BNE .7
088D- E6 07    2370      INC A.PNTR+1
088F- A5 07    2380      .7      LDA A.PNTR+1
0891- C9 55    2390      CMP /ARRAY+SIZE
0893- 90 AA    2400      BCC .2
0895- 60      2410      RTS
2420 *
2430 * OPTIONAL PRINT PRIME SUBROUTINE
2440 *
0896- A4 1C    2450      PRINTP LDY PRIME+1      HI BYTE
0898- A6 1B    2460      LDX PRIME
089A- 20 40 F9 2470      JSR PRINTN      PRINT DECIMAL VAL
089D- 20 9E FD 2480      JSR LINE      VIDEO "-" OUT
08A0- 60      2490      RTS

```

Michael R. Laumer, of Carrollton, Texas, has been working for about a year on a full-scale compiler for the Integer BASIC language. He has it nearly finished now, so just for fun he used it to compile the algorithm from Gilbreath's article. Mike used a slightly different form of the Integer BASIC program than I did, which took 238 seconds to execute. But the compiled version ran in only 20 seconds! If you are interested in compiling Integer BASIC programs, you can write to Mike at Laumer Research, 1832 School Road, Carrollton, TX 75006.

If you want to, you can easily cut the time of my program from .74 to about .69 seconds. Lines 1600-1650 in my program set each byte in ARRAY to \$01. If I don't mind the extra program length, I can rewrite this loop to run in about 42 milliseconds instead of the over 90 it now takes. Here is how I would do it:

```
.1      STA ARRAY,Y
        STA ARRAY+$100,Y
        STA ARRAY+$200,Y
        STA ARRAY+$300,Y
        .
        .
        .
        STA ARRAY+$1E00,Y
        STA ARRAY+$1F00,Y
        INY
        BNE .1
```

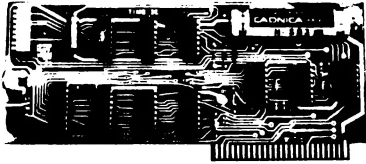
TOTAL OF 32
LINES LIKE THESE

If you can find a way to implement the same program in less than .69 seconds, you are hereby challenged to do so!

Time II

The most powerful, easiest to use, clock for your APPLE

- TIME IN HOURS, MINUTES AND SECONDS.
- DATE WITH YEAR, MONTH, DATE, DAY OF WEEK AND LEAP YEAR.
- FAST DATE AND TIME SETTING.
- PROGRAM SELECTABLE 24 HOUR MILITARY FORMAT OR 12 HOUR WITH AM/PM FORMAT.
- ± 30 SECOND ADJUST.
- DIP SWITCH SELECTABLE INTERRUPTS PERMIT FOREGROUND/BACKGROUND OPERATION OF TWO PROGRAMS SIMULTANEOUSLY SO YOU CAN CALL UP SCHEDULES, TIME EVENTS, DATE LISTINGS, AND OTHER PRINTOUTS.
- CRYSTAL CONTROLLED FOR .0005% ACCURACY.
- LATCHED INPUT AND OUTPUT PORTS FOR THE EASIEST PROGRAMMING IN BASIC.
- ON BOARD BATTERY BACKUP POWER FOR OVER 4 MONTHS POWER OFF OPERATION (BATTERY CHARGES WHEN APPLE IS ON).




ALL ORDERS SHIPPED SAME DAY
SEND \$129.00 CHECK OR MONEY ORDER
(TEXAS RESIDENTS ADD 5% SALES TAX)

APPLIED ENGINEERING
P.O. BOX 470301
DALLAS, TEXAS 75247


• INCLUDES 16 SECTOR DISK WITH OVER 25 CONTRIBUTED PROGRAMS SO YOU CAN PUT YOUR TIME II TO USE RIGHT AWAY.

• TWENTY-THREE PAGE OPERATING MANUAL INCLUDED, WITH MANY EXAMPLES OF PROGRAMS TO USE WITH YOUR APPLE IN ANY CONFIGURATION.

MASTER CHARGE & VISA WELCOME



(214) 492-2027



7:00 AM - 11:00 PM 7 DAYS A WEEK
APPLE PERIPHERALS ARE OUR ONLY BUSINESS

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203
817/382-6353

DIS-ASSEMBLER

DSA-DS dis-assembles Apple machine language programs into forms compatible with LISA, S-C ASSEMBLER (3.2 or 4.0), Apple's TOOL-KIT ASSEMBLER and others. DSA-DS dis-assembles instructions or data. Labels are generated for referenced locations within the machine language program.

\$25, Disk, Applesoft (32K, ROM or Language card)

OTHER PRODUCTS

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.

\$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE...**, **CASE**, **SELECT**, and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.

\$35. Disk, Applesoft (48K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output forms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.

\$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.

\$25 Disk, Applesoft.

SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.

\$15 Disk, Applesoft (32K, ROM or Language Card).

(Add \$4.00 for Foreign Mail)

*Apple II is a registered trademark of the Apple Computer Co.

6809 Cross Assembler

Chris Wiggs, of Rockford, IL, has developed a cross assembler for the 6809 which runs in the Apple. In fact, it is really a set of patches to the S-C Assembler II Version 4.0. If you BLOAD your copy of the assembler, and then BRUN his patch file, and BSAVE the result, you have a brand new assembler for 6809 code.

It is set up to work with "The Mill". Typing MGO turns on the mill and starts 6809 code executing, while the Apple's 6502 is left in a waiting loop.

Chris has authorized me to distribute these patches. For only \$20 you will get a disk which includes all of the source code for the patches (in S-C Assembler II Version 4.0 format), the already-assembled patch file, a sample 6809 program, and some instructions (in the form of an assembly source file of comments).

I have not put this program through any rigorous test, but Chris is using it himself and is satisfied that it is working correctly. Anyway, you will actually have the SOURCE code, so you can make any further changes you wish with ease.

You might also study how he did it, and then write a cross assembler for some other chip, such as Z-80, 68000, 1802, TMS7000, or whatever.

Here is a sample 6809 assembly:

```

1000 *-----
1010 *      6809 MULTI-PRECISION ADDITION SUBROUTINE
1020 *      FROM "6809 ASSEMBLY LANGUAGE PROGRAMMING",
1030 *      LANCE LEVENTHAL, OSBORNE/MCGRAW-HILL,
1040 *      PAGE 11-7
1050 *-----
1060 *      CALL: JSR MPADD
1070 *      .DA #N      NUMBER OF BYTES TO ADD
1080 *      .DA ARG1    ADDRESS OF FIRST ARGUMENT
1090 *      .DA ARG2    ADDRESS OF 2ND ARGUMENT
1100 *      .DA SUM     ADDRESS FOR SUM
1110 *-----
0800- 34 77 1120 MPADD PSHS X,Y,U,A,B,CCR SAVE ALL REGISTERS
0802- EE 69 1130 LDU 9,S ACCESS PARAMETER LIST
0804- 37 34 1140 PULU X,Y,B GET LENGTH AND ADDRESSES OF ARGS
0806- EE C4 1150 LDU ,U GET ADDRESS OF SUM
0808- 1C FE 1160 ANDCC #SFE CLEAR CARRY TO START SUM
080A- A6 80 1170 .1 LDA ,X+ GET BYTE FROM 1ST ARG
080C- A9 A0 1180 ADCA ,Y+ ADD BYTE FROM 2ND ARG
080E- A7 C0 1190 STA ,U+ STORE BYTE IN SUM
0810- 5A 1200 DECB ALL BYTES ADDED?
0811- 26 F7 1210 BNE .1 NOT YET
0813- EE 69 1220 LDU 9,S ADJUST RETURN ADDRESS PAST
0815- 33 47 1230 LEAU 7,U THE ARGUMENT LIST
0817- EF 69 1240 STU 9,S
0819- 35 F7 1250 PULS PC,U,Y,X,B,A,CCR RESTORE REGISTERS, RETURN

```

SYMBOL TABLE

```

0800- MPADD
.01=080A

```

JOHN'S DEBUGGER & DISASSEMBLER

FOR
ASSEMBLY LANGUAGE PROGRAMMING
ON THE APPLE II COMPUTER

NOW YOU CAN **TRACE** OR **STEP** ANY 6502 INSTRUCTION
LOCATED ANYWHERE IN MEMORY

BEGIN DEBUGGING FROM **ANY POINT** WITHIN YOUR PROGRAM
COMPUTES EFFECTIVE ADDRESS FOR ALL ADDRESSING MODES
& DISPLAYS ALL MEMORY CHANGES (BEFORE/AFTER)
Options to quickly move thru your program-selected
memory, equal zero, leave subroutine, etc

TRACE LOGIC W/ INSTRUCTION - NOTING ALL Jmps,JSR, etc

STEP EACH INSTRUCTION DISPLAYING:

-ALL REGS, STATUS & POINTER -ACCUMULATOR IN BINARY
-LAST 8 BYTES ON THE STACK -DISPLAY OF ALL FLAGS SET
-DISPLAY WHAT IS IN ANY 12 MEMORY POSITIONS WITH LABELS
OPTIONS CAN BE USED IN ANY ORDER: STEP, TRACE, CONTINUOUS,
ONE PAGE, SINGLE LINE, MONITOR EXIT & RETURN TO PROCESSING

BREAKPOINT BREAK ON KEYPRESS, CYCLE COUNTER,ETC
(6 OPTIONS) INCLUDES TIMING DELAY FROM 0.0 to 200 SECONDS
(ALL OF THE ABOVE SAVES THE ENTIRE PAGE OF THE STACK)
REQUIRES: 48K (MACH LANGUAGE USES FROM 8400 TO 9600)

JOHN'S DEBUGGER..\$ 49.95 JOHN'S DISASSEMBLER \$ 29.95

Disassembler can disassemble into both ASC II

& 6502 instruction set simultaneously

----- BOTH ON DISKETTE FOR \$ 59.95 -----

JOHN BRODERICK, CPA
BRODERICK & ASSOCIATES
8635 SHAGROCK
DALLAS, TEXAS 75238

Extending the Apple Monitor

Just as the creators of Applesoft included the wonderful "&" statement to allow language extensions, so also Steve Wozniak included a means for adding new monitor commands. The "control-Y" command branches to a user-defined machine language routine, which can supplement the existing commands in the Monitor ROM.

The control-Y command executes your subroutine starting at \$3F8. All there is room for at \$3F8 is a JMP to where your subroutine is REALLY stored. When you boot DOS, a JMP \$FF65 instruction is inserted at \$3F8, setting the control-Y command to merely re-enter the monitor. By changing the address of that JMP instruction, you can have it jump to your own code. If you look ahead at the listing of MONITOR EXTENSIONS, lines 1170-1210 store the address of my CTRL_Y subroutine into the JMP instruction.

I have thought of at least three features that I miss all the time in the monitor. (I just now thought of several more, but they will have to wait for another article.)

1. The monitor already includes the ability to add and subtract single-byte values, and print the single-byte result. I would like to be able to do this with 16-bit values.
2. The monitor can already dump memory in hexadecimal, but I want to see it as ASCII characters also. There is room on the screen for both at once.
3. The monitor can already disassemble code to the screen, 20 lines at a time. If I want more than 20 lines, I can type "LLLLLL", one L for each 20 lines. But I would like to be able to just specify the beginning and ending addresses for the disassembly, like I do for the hexadecimal printout.

If you enter the MONITOR EXTENSIONS program, these three functions will be added to the monitor. To add or subtract two values, type the two values separated by "+" or "-"; then type control-Y, and carriage return. To dump in combined hex and ASCII, type the beginning and ending addresses separated by a period, then control-Y and carriage return. To disassemble a range of memory, type the beginning and ending addresses separated by a period, then control-Y, "L", and a carriage return.

Looking again at the listing, lines 1230-1340 figure out which of the above command options you have typed in. When the monitor branches to \$3F8, the following conditions have been set up:

- (A) = 0 if only one address was typed;
 = code for separator character if two addresses
 were typed.

(X) = 0 if no hex digit typed immediately before the control-Y;
= 1 if any hex digits immediately before the control-Y.

(Y) = 0

(\$34) = index into input buffer of next character after the control-Y.

Up to five 16-bit variables (called A1, A2, A3, A4, and A5) are filled from the hexadecimal values in the command. If you type a "<" after the first value, then that value will be stored in A4 and A5 (A4 is at \$42,43; A5 at \$44,45). If you type a ".", "+", "-", or ":" after a hexadecimal value, then that value will be stored in A1 and A3 (A1 is at \$3C,3D; A3 at \$40,41). If you type a hexadecimal value immediately before the control-Y, then that value will be stored in A2 (which is at \$3E,3F).

Looking again at lines 1230-1340, I branch to SUB if the separator is "-", or ADD if it is "+". If the separator is a colon, I just return; I don't have any control-Y command which accepts a colon separator. If the separator is not any of the above, then either there was no separator, or it was a period. In both of these cases, I want to dump memory. If the character after the control-Y is not "L", then I want a combined hex-ASCII dump; if it is "L", I want disassembly. Line 1340 increments the buffer pointer so that the "L" command will not be re-executed by the regular monitor routine after my control-Y routine is finished.

Lines 1360-1450 control the disassembly option. I used a monitor subroutine to copy the beginning address from A1 into PC. Then I wrote a loop that calls the monitor routine to disassemble one line, and then checks to see if we have reached the ending address. Compare this to the code in the monitor ROM at \$FE5E through \$FE74. There is one trick in this code. I wanted to compare PC to END.ADDR, and continue if PC was less than or equal to END.ADDR. The normal comparison technique would either SET carry at line 1390, but I CLEARED it. This has the same affect as using one less than the value in PC as the first comparand. I needed this, because BCC at line 1440 only branches if the first comparand is LESS THAN the second one. In other words, since it is difficult to implement IF PC <= END.ADDR THEN ..., I implemented IF PC-1 < END.ADDR THEN

Lines 1470-1780 perform the combined hex-ASCII dump. I must give credit to Hugh McKinney, of Dunwoody, GA, for some of the ideas in this code. Just for fun, I set it up to always print complete rows of eight bytes; the starting address is rounded down to the nearest multiple of 8, and the ending address is rounded up. This means that typing just one address will get you eight, also.

I had to make a judgment about what characters to display for the ASCII portion of the dump. There are 256 possible values, and only 96 printing characters. In fact, if you don't have a lower

case adapter, your screen only shows 64 printing characters (unless you count inverse and flashing characters as different; in that case you have 192). I decided to display control characters (codes 00-1F and 80-9F) as flashing characters (codes 40-5F). Codes 60-7F and E0-FF display as lower case characters if you have a lower case adapter. Codes 20-5F and A0-DF display as normal video characters (the standard upper case set). If you want a different mapping, change lines 1660-1690 to do it your way.

Lines 1800-1930 perform the 16-bit addition and subtraction in the normal way. Lines 1940-1980 print out an equal sign, and the value.

If you get really ambitious, you might try programming for your Apple II Plus the S and T commands that Apple removed from the Autostart ROM. You can just about copy the code right out of the reference manual. You might also like to add a memory move command that will work correctly even when the target area overlaps the source area.

```

1000 *-----
1010 *      MONITOR EXTENSIONS
1020 *-----
0034- 1030 MON.YSAV .EQ $34
003A- 1040 PC .EQ $3A,3B
003C- 1050 BGN.ADDR .EQ $3C,3D
003E- 1060 END.ADDR .EQ $3E,3F
0200- 1070 WBUF .EQ $200
F940- 1080 MON.PRNTYX .EQ $F940
FCBA- 1090 MON.NXTAL .EQ $FCBA
FDA3- 1100 MON.XAM8 .EQ $FDA3
FDED- 1110 MON.COUT .EQ $FDED
FE63- 1120 MON.LIST .EQ $FE63
FE75- 1130 MON.ALPC .EQ $FE75
1140 *-----
1150 *      .OR $300
1160 *-----
0300- A9 0B 1170 SETUP LDA #CTRLY
0302- 8D F9 03 1180 STA $3F9
0305- A9 03 1190 LDA /CTRLY
0307- 8D FA 03 1200 STA $3FA
030A- 60 1210 RTS
1220 *-----
030B- C9 AD 1230 CTRLY CMP #SAD MINUS?
030D- F0 6B 1240 BEQ SUB
030F- C9 AB 1250 CMP #SAB PLUS?
0311- F0 74 1260 BEQ ADD
0313- C9 BA 1270 CMP #SBA COLON?
0315- F0 20 1280 BEQ RETURN
0317- A4 34 1290 LDY MON.YSAV LOOK BEYOND CONTROL-Y
0319- B9 00 02 1300 LDA WBUF,Y
031C- A0 00 1310 LDY #0
031E- C9 CC 1320 CMP #'L+$80
0320- D0 16 1330 BNE DUMP
0322- E6 34 1340 INC MON.YSAV
1350 *-----
0324- 20 75 FE 1360 DISASM JSR MON.ALPC
0327- A9 01 1370 .1 LDA #1 DISASSEMBLE ONE LINE
0329- 20 63 FE 1380 JSR MON.LIST
032C- 18 1390 CLC
032D- A5 3A 1400 LDA PC
032F- E5 3E 1410 SEC END.ADDR
0331- A5 3B 1420 LDA PC+1
0333- E5 3F 1430 SEC END.ADDR+1
0335- 90 F0 1440 BCC .1
0337- 60 1450 RETURN RTS
1460 *-----
0338- A5 3E 1470 DUMP LDA END.ADDR
033A- 09 07 1480 ORA #7 FINISH LAST ROW OF 8
033C- 85 3A 1490 STA PC
033E- A5 3F 1500 LDA END.ADDR+1
0340- 85 3B 1510 STA PC+1

```


0342-	A5	3C	1520	LDA	BGN,ADDR	START WITH FULL ROW OF 8
0344-	29	F8	1530	AND	#F8	
0346-	85	3C	1540	STA	BGN,ADDR	
0348-	20	A3	FD 1550	JSR	MON.XAM8	
034B-	38		1560	SEC		BACK UP POINTER FOR ROW
034C-	A5	3C	1570	LDA	BGN,ADDR	
034E-	E9	08	1580	SEC	#8	
0350-	85	3C	1590	STA	BGN,ADDR	
0352-	B0	02	1600	BCS	.2	NO BORROW
0354-	C6	3D	1610	DEC	BGN,ADDR+1	
0356-	A9	A0	1620	LDA	#SA0	PRINT BLANK
0358-	20	ED	FD 1630	JSR	MON.COUT	
035B-	A0	00	1640	LDY	#0	
035D-	B1	3C	1650	LDA	(BGN,ADDR),Y	
035F-	09	80	1660	ORA	#S80	MAKE NORMAL VIDEO
0361-	C9	A0	1670	CMP	#SA0	SEE IF PRINTABLE
0363-	B0	02	1680	BCS	.4	YES
0365-	49	C0	1690	BCR	#SC0	MAKE CONTROLS INTO FLASHING ALPHA
0367-	20	ED	FD 1700	JSR	MON.COUT	PRINT IT
036A-	20	BA	FC 1710	JSR	MON.NXTAI	ADVANCE POINTER
036D-	90	EC	1720	BCC	.3	MORE ON THIS ROW
036F-	A5	3C	1730	LDA	BGN,ADDR	
0371-	C5	3A	1740	CMP	PC	SEE IF FINISHED WITH DUMP
0373-	A5	3D	1750	LDA	BGN,ADDR+1	
0375-	E5	3B	1760	SEC	PC+1	
0377-	90	CF	1770	BCC	.1	NO
0379-	60		1780	RTS		YES
			1790	*		
037A-	38		1800	SUB	SEC	
037B-	A5	3C	1810	LDA	BGN,ADDR	
037D-	E5	3E	1820	SEC	END,ADDR	
037F-	AA		1830	TAX		
0380-	A5	3D	1840	LDA	BGN,ADDR+1	
0382-	E5	3F	1850	SEC	END,ADDR+1	
0384-	4C	91	03 1860	JMP	AS1	
			1870	*		
0387-	18		1880	ADD	CLC	
0388-	A5	3C	1890	LDA	BGN,ADDR	
038A-	65	3E	1900	ADC	END,ADDR	
038C-	AA		1910	TAX		
038D-	A5	3D	1920	LDA	BGN,ADDR+1	
038F-	65	3F	1930	ADC	END,ADDR+1	
0391-	A8		1940	AS1	TAY	
			1950			
0392-	A9	BD	1960	LDA	#SBD	EQUAL SIGN
0394-	20	ED	FD 1970	JSR	MON.COUT	
0397-	4C	40	F9 1980	JMP	MON.PRINTYX	



THE KEY TO UNDERSTANDING THE APPLE DOS 3.3...

**COMPUTER DATA SERVICES proudly announces Lazer Systems
DOSOURCE 3.3. A source listing of DOS 3.3**

Randy Hyde has disassembled DOS 3.3, added meaningful labels and comments and came up with **DOSOURCE 3.3**.

DOSOURCE 3.3 clearly lists all the routines used within Apple's DOS and removes the mystery that surrounded DOS 3.3 until now. Never before have the internals of DOS 3.3 been so explicitly explained. Now professional programmers, hackers, and the curious can really see what goes on inside DOS 3.3.

DOSOURCE 3.3 comes on two diskettes (four sides) in three formats:

- LISA 2.5 compatible source listing that can be reassembled by LISA owners.
- text file listing that can be converted for use by other assemblers.
- assembled listing showing all the addresses and hex values with DOS.

With **DOSOURCE 3.3** you can:

- Reassemble DOS at different addresses (for example, the **Andromeda 16K RAM** board or language card).
- Optimize portions of DOS.

- Utilize those useful 'mystery' routines found within DOS.
- Remove unneeded portions of DOS for application programs.
- Learn all kinds of neat programming tricks.
- Write your own DOS once you see how Apple did it.
- Become a DOS expert.
- Write useful DOS utility programs. CDS will assist you in marketing programs you've written for Apple DOS.

Special Introductory Price: \$39.95 (check, COD, VISA / MASTERCARD)

Lazer Systems Lower Case Plus, Keyboard Plus, and **DOSOURCE 3.3** are all available from **COMPUTER DATA SERVICES**.



P.O. Box 696, Amherst, NH 03031 (603) 673-7375

Errata

Volume 1, Issue 12 (Sep 1981) page 8: Line 1120 in the CHRGET/CHRGOT subroutine should be BCS instead of BEQ.

Volume 1, Issue 7 (Apr 1981) page 8: Insert the following lines:

```
1331      TXA          LINE LENGTH
1332      TAY          IN Y-REG FOR LOOP COUNT
1333 .2    LDA $200,Y  STRIP SIGN-BITS FROM EACH BYTE
1334      AND #$7F
1335      STA $200,Y
1336      DEY
1337      BPL .2
```

This patch is necessary because charactersⁿ in Applesoft strings are supposed to have the sign-bit clear. Everything is fine unless you try compare input strings with constant strings.

DOS Disassembly: \$B052-B0B5 AND \$B35F-B7FF

Everything from \$B800 through \$BFFF has now been covered in previous issues of AAL. Also, the 3.3 boot ROM was covered in the August issue. In this issue I present the rest of the boot code and part of the File Manager (FM).

Lines 1000-1570 are a subroutine inside FM which calls RWTS. The main entry at line 1170 assumes (A)=opcode, (X)=track, and (Y)=sector. A subsidiary entry at line 1200 assumes (A)=opcode, and track and sector were already set up. The valid opcodes are SEEK=0, READ=1, WRITE=2, and FORMAT=4.

Lines 1580-1970 are the various exits from FM. Upon exit, (A)=error code and CARRY status is set if there was an error, clear if not.

Lines 1980-2560 are various buffers, constants, and variables for FM. Notice there are some apparently unused bytes in this area.

Lines 2570-3690 are what is written on track 0 sector 0. It loads and executes BOOT.STAGE1 at \$0800 (execution starts at \$0801). This code reads in RWTS and BOOT.STAGE2. Since most of this area was unused, patches to solve the APPEND problem are here (lines 3020-3640).

Lines 3700-4080 are BOOT.STAGE2, which read in the rest of DOS and jump to \$9D84.

Routines to write the DOS image on tracks 0-2, to enter RWTS with interrupts disabled, and to clear a 256-byte buffer are in lines 4090-4990.

Lines 5100-5300 are the IOB and DCT used by FM for all calls to RWTS. The contents of these are described in the DOS Reference Manual pages 95-98.

```

1000 *
1010 *      DOS 3.2.1/3.3 FILE MANAGER $B052-B0B5
1020 *
1030      .OR $B052
1040      .TA $0852
1050 *
0048- 1060 MON.STATUS .EQ $48
AAC1- 1070 IOB.ADDR .EQ $AAC1
AE7E- 1080 SAVE.FMW .EQ $AE7E
BD00- 1090 RWTS .EQ $BD00
FB2F- 1100 MON.INIT .EQ $FB2F
FC58- 1110 MON.HOME .EQ $FC58
FDDA- 1120 MON.PRVYTE .EQ $FDDA
FDED- 1130 MON.COUT .EQ $FDED
FE89- 1140 MON.SETKBD .EQ $FE89
FE93- 1150 MON.SETVID .EQ $FE93
1160 *
1170 CALL.RWTS
B052- 8E EC B7 1180 STX IOB.TRACK
B055- 8C ED B7 1190 STX IOB.SECTOR
1200 CALL.RWTS.1
B058- 8D F4 B7 1210 STA IOB.OPCODE (SEEK=0, READ=1, WRITE=2, FORMAT=4)
B05B- C9 02 1220 CMP #2 OPCODE="WRITE"?
B05D- D0 06 1230 BNE .1
B05F- 0D D5 B5 1240 ORA FMW.FLAGS SET "LAST OP WAS WRITE" FLAG
B062- 8D D5 B5 1250 STA FMW.FLAGS
B065- AD F9 B5 1260 .1 LDA FMW.VOLUME
B068- 49 FF 1270 BOR #$FF UN-COMPLEMENT THE VOLUME #
B06A- 8D EB B7 1280 STA IOB.VOLUME
B06D- AD F7 B5 1290 LDA FMW.SLOT16 SLOT # TIMES 16
B070- 8D E9 B7 1300 STA IOB.SLOT16
B073- AD F8 B5 1310 LDA FMW.DRIVE DRIVE #
B076- 8D EA B7 1320 STA IOB.DRIVE
B079- AD E2 B5 1330 LDA FMW.SECTSZ SECTOR LENGTH IN BYTES
B07C- 8D F2 B7 1340 STA IOB.SECTSZ
B07F- AD E3 B5 1350 LDA FMW.SECTSZ+1
B082- 8D F3 B7 1360 STA IOB.SECTSZ+1
B085- A9 01 1370 LDA #1 SET TABLE TYPE
B087- 8D E8 B7 1380 STA IOB.TYPE
B08A- AC C1 AA 1390 LDY IOB.ADDR GET ADDRESS OF IOB
B08D- AD C2 AA 1400 LDA IOB.ADDR+1
B090- 20 B5 B7 1410 JSR ENTER.RWTS PERFORM THE OPERATION
B093- AD F6 B7 1420 LDA IOB.ACTVOL VOLUME # FOUND
B096- 8D BF B5 1430 STA FMW.DATA+2
B099- A9 FF 1440 LDA #$FF RESET VOLUME EXPECTED IN IOB
B09B- 8D EB B7 1450 STA IOB.VOLUME
B09E- B0 01 1460 BCS .2 CARRY SET IF RWTS ERROR
B0A0- 60 1470 RTS RETURN TO CALLER
B0A1- AD F5 B7 1480 .2 LDA IOB.ERROR GET ERROR CODE
B0A4- A0 07 1490 LDY #7 ERR=7 IF VOLUME MISMATCH
B0A6- C9 20 1500 CMP #$20 VOLUME MISMATCH?
B0A8- F0 08 1510 BEQ .3 YES
B0AA- A0 04 1520 LDY #4 ERR=4 IF WRITE PROTECTED
B0AC- C9 10 1530 CMP #$10 WRITE PROTECTED?
B0AE- F0 02 1540 BEQ .3 YES
B0B0- A0 08 1550 LDY #8 ERR=8 (I/O ERROR) FOR ALL OTHERS
B0B2- 98 1560 .3 TYA ERR IN A-REG
B0B3- 4C 85 B3 1570 JMP FM.EXIT.ERROR
1580 *
1590 *      DOS 3.3 FILE MANAGER $B35F-B5FF
1600 *
1610      .OR $B35F
1620      .TA $0B5F
B35F- A9 01 1630 FM.EXIT.ERR1 LDA #1 "LANGUAGE NOT AVAILABLE"
B361- D0 22 1640 BNE FM.EXIT.ERROR
B363- A9 02 1650 FM.EXIT.ERR2 LDA #2 "RANGE ERROR" (OPCODE)
B365- D0 1E 1660 BNE FM.EXIT.ERROR
B367- A9 03 1670 FM.EXIT.ERR3 LDA #3 "RANGE ERROR" (SUBCODE)
B369- D0 1A 1680 BNE FM.EXIT.ERROR
B36B- A9 04 1690 FM.EXIT.ERR4 LDA #4 "WRITE PROTECTED"
B36D- D0 16 1700 BNE FM.EXIT.ERROR
B36F- A9 05 1710 FM.EXIT.ERR5 LDA #5 "END OF DATA"
B371- D0 12 1720 BNE FM.EXIT.ERROR
B373- A9 06 1730 FM.EXIT.ERR6 LDA #6 "FILE NOT FOUND"
B375- D0 0E 1740 BNE FM.EXIT.ERROR
B377- 4C ED BF 1750 FM.EXIT.ERR9 JMP $BFED "DISK FULL"
B37A- EA 1760 NOP
B37B- A9 0A 1770 FM.EXIT.ERR10 LDA #10 "FILE LOCKED"
B37D- D0 06 1780 BNE FM.EXIT.ERROR

```

		1790	*-----	
		1800	FM.EXIT.GOOD	
B37F-	AD C5 B5	1810	LDA FMP.RETURN	GET RETURN CODE (ZERO)
B382-	18	1820	CLC	SIGNAL NO ERROR
B383-	90 01	1830	BCC FM.EXIT	...ALWAYS
		1840	*-----	
		1850	FM.EXIT.ERROR	
B385-	38	1860	SEC	
		1870	*-----	
		1880	FM.EXIT	
B386-	08	1890	PHP	SAVE STATUS ON STACK
B387-	8D C5 B5	1900	STA FMP.RETURN	RETURN CODE
B38A-	A9 00	1910	LDA #0	CLEAR MONITOR STATUS (JUST IN CASE)
B38C-	85 48	1920	STA MON.STATUS	
B38E-	20 7E AE	1930	JSR SAVE.FMW	SAVE FM WORKAREA IN FILE BUFFER
B391-	28	1940	PLP	RETRIEVE STATUS FROM STACK
B392-	AE 9B B3	1950	LDX FMS.STACK	RESTORE STACK POINTER
B395-	9A	1960	TXS	
B396-	60	1970	RTS	RETURN TO WHOEVER CALLED FM
		1980	*-----	
		1990	* SCRATCH AREA	
		2000	*-----	
B397-		2010	FMS.TS.CD .BS 2	T/S OF CURRENT DIRECTORY SECTOR
B399-		2020	.BS 2	?
B39B-		2030	FMS.STACK .BS 1	S-REG WHEN FM CALLED
B39C-		2040	FMS.DIRNDX .BS 1	VARIOUS USES
B39D-		2050	.BS 1	
B39E-		2060	.BS 2	?
B3A0-	00 00 FF			
B3A3-	FF	2070	.HS 0000FFFF USED BY INIT TO CLEAR VTOC ENTRY	
		2080	*-----	
B3A4-	01 0A 64	2090	.DA #1,#10,#100 DECIMAL CONVERSION TABLE	
B3A7-	D4 C9 C1			
B3AA-	C2 D3 D2			
B3AD-	C1 C2	2100	.AS -/TIABSRAB/ FILE TYPE CODES	
B3AF-	A0 C5 CD			
B3B2-	D5 CC CF			
B3B5-	D6 A0 CB			
B3B8-	D3 C9 C4	2110	.AS -/ EMULOV KSID/ MSG SPELLED BACKWARDS	
		2120	*-----	
		2130	* VTOC SECTOR BUFFER	
		2140	*-----	
B3BB-		2150	.BS 256	
		2160	*-----	
		2170	* DIRECTORY SECTOR BUFFER	
		2180	*-----	
B4BB-		2190	.BS 256	
		2200	*-----	
		2210	* FILE MANAGER PARAMETERS	
		2220	*-----	
B5BB-		2230	FMP.OPCODE .BS 1	
B5BC-		2240	FMP.SUBCOD .BS 1	
B5BD-		2250	FMP.DATA .BS 8	USE DEPENDS ON OPCODE
B5C5-		2260	FMP.RETURN .BS 1	ERROR CODE
B5C6-		2270	.BS 1	?
B5C7-		2280	FMP.PNTR.WORK .BS 2	ADDR OF WORKAREA IN FILE BUFFER
B5C9-		2290	FMP.PNTR.TS .BS 2	ADDR OF T/S LIST IN FILE BUFFER
B5CB-		2300	FMP.PNTR.DATA .BS 2	ADDR OF DATA IN FILE BUFFER
B5CD-		2310	.BS 4	?
		2320	*-----	
		2330	* FILE MANAGER WORKAREA	
		2340	*-----	
B5D1-		2350	FMW.TS.TS1 .BS 2	T/S OF FIRST T/S LIST SECTOR
B5D3-		2360	FMW.TS.TSC .BS 2	T/S OF CURRENT T/S LIST SECTOR
B5D5-		2370	FMW.FLAGS .BS 1	CHECKPOINT FLAGS
B5D6-		2380	FMW.TS.DATA .BS 2	T/S OF CURRENT DATA SECTOR
B5D8-		2390	.BS 2	DIRECTORY SECTOR INDEX
B5DA-		2400	.BS 2	# SECTORS PER TS LIST
B5DC-		2410	.BS 2	1ST SECTOR
B5DE-		2420	.BS 2	LAST SECTOR+1
B5E0-		2430	.BS 2	CURRENT SECTOR
B5E2-		2440	FMW.SECTSZ .BS 2	SECTOR SIZE IN BYTES
B5E4-		2450	.BS 4	FILE POSITION
B5E8-		2460	.BS 2	RECORD LENGTH FROM OPEN
B5EA-		2470	.BS 2	RECORD NUMBER
B5EC-		2480	.BS 2	BYTE OFFSET INTO RECORD
B5EE-		2490	.BS 2	# SECTORS IN FILE
B5F0-		2500	.BS 6	SECTOR ALLOCATION AREA
B5F6-		2510	FMW.FILTYP .BS 1	
B5F7-		2520	FMW.SLOT16 .BS 1	

```

B5F8- 2530 FMW.DRIVE .BS 1
B5F9- 2540 FMW.VOLUME .BS 1 (COMPLEMENT FORM)
B5FA- 2550 FMW.TRACK .BS 1
B5FB- 2560 .BS 5 <NOT USED>
2570 *
2580 * STAGE 1 OF BOOT (EXECUTES AT $0800)
2590 *
2600 .OR $800
2610 .TA $E00
2620 BOOT.STAGE1
2630 .HS 01
0800- 01 2640 * COMES HERE AFTER EACH SECTOR IS READ
0801- A5 27 2650 LDA $27 NEXT PAGE TO READ INTO
0803- C9 09 2660 CMP #9 FIRST TIME HERE?
0805- D0 18 2670 BNE .1 NO, SKIP OVER INITIALIZATION
0807- A5 2B 2680 LDA $2B SLOT*16
0809- 4A 2690 LSR GET SLOT #
080A- 4A 2700 LSR
080B- 4A 2710 LSR
080C- 4A 2720 LSR
080D- 09 C0 2730 ORA #$C0 BUILD ADDRESS INTO ROM
080F- 85 3F 2740 STA $3F FOR READING A SECTOR
0811- A9 5C 2750 LDA #$5C
0813- 85 3E 2760 STA $3E
0815- 18 2770 CLC
0816- AD FE 08 2780 LDA BT1.ADDR+1 COMPUTE ADDRESS OF LAST PAGE
0819- 6D FF 08 2790 ADC BT1.N TO BE READ
081C- 8D FE 08 2800 STA BT1.ADDR+1
081F- AE FF 08 2810 .1 LDX BT1.N # PAGES LEFT TO READ - 1
0822- 30 15 2820 BMI .2 FINISHED
0824- BD 4D 08 2830 LDA SECTOR.NUMBER,X CONVERT TO PHYSICAL SECTOR #
0827- 85 3D 2840 STA $3D
0829- CE FF 08 2850 DEC BT1.N
082C- AD FE 08 2860 LDA BT1.ADDR+1
082F- 85 27 2870 STA $27
0831- CE FE 08 2880 DEC BT1.ADDR+1
0834- A6 2B 2890 LDX $2B SLOT*16
0836- 6C 3E 00 2900 JMP ($3E) READ NEXT SECTOR
0839- EE FE 08 2910 .2 INC BT1.ADDR+1 POINT AT STAGE 2 LOADER
083C- EE FE 08 2920 INC BT1.ADDR+1
083F- 20 89 FE 2930 JSR MON.SETKBD
0842- 20 93 FE 2940 JSR MON.SETVID
0845- 20 2F FB 2950 JSR MON.INIT
0848- A6 2B 2960 LDX $2B SLOT*16
084A- 6C FD 08 2970 JMP (BT1.ADDR)
2980 *
2990 SECTOR.NUMBER
084D- 00 0D 0B
0850- 09 07 05
0853- 03 01 3000 .HS 00D0B0907050301
0855- 0E 0C 0A
0858- 08 06 04
085B- 02 0F 3010 .HS 0E0C0A080604020F
3020 *
3030 * DOS 3.3 PATCHES FOR APPEND AND VERIFY
3040 *
3050 .OR $B65D
3060 .TA $0E5D
B65D- 3070 APPEND.FLAG .BS 1
3080 PATCH.DOS33.1
B65E- 20 64 A7 3090 JSR $A764 LOCATE AND FREE FILE BUFFER
B661- B0 08 3100 BCS .1
B663- A9 00 3110 LDA #0 CLEAR APPEND FLAG
B665- A8 3120 TAY
B666- 8D 5D B6 3130 STA APPEND.FLAG
B669- 91 40 3140 STA ($40),Y
B66B- AD C5 B5 3150 .1 LDA FMP.RETURN
B66E- 4C D2 A6 3160 JMP $A6D2
3170 *
3180 PATCH.DOS33.2
B671- AD 5D B6 3190 LDA APPEND.FLAG
B674- F0 08 3200 BEQ .1
B676- EE BD B5 3210 INC FMP.DATA
B679- D0 03 3220 BNE .1
B67B- EE BE B5 3230 INC FMP.DATA+1
B67E- A9 00 3240 .1 LDA #0 CLEAR APPEND FLAG
B680- 8D 5D B6 3250 STA APPEND.FLAG
B683- 4C 46 A5 3260 JMP $A546

```

```

3270 *-----
3280 PATCH.DOS33.3
B686- 8D BC B5 3290 STA FMP.SUBCOD
B689- 20 A8 A6 3300 JSR $A6A8
B68C- 20 EA A2 3310 JSR $A2EA
B68F- 4C 7D A2 3320 JMP $A27D
3330 *-----
3340 PATCH.DOS33.4
B692- A0 13 3350 LDY #19 LOOK AT FILE POSITION
B694- B1 42 3360 .1 LDA ($A2),Y
B696- D0 14 3370 BNE .4 NOT AT 0000
B698- C8 3380 INY
B699- C0 17 3390 CPY #23
B69B- D0 F7 3400 BNE 1 TEST 4 BYTES
B69D- A0 19 3410 LDY #25
B69F- B1 42 3420 .2 LDA ($A2),Y
B6A1- 99 A4 B5 3430 STA FMP.DATA-25,Y
B6A4- C8 3440 INY
B6A5- C0 1D 3450 CPY #29 MOVE 4 BYTES
B6A7- D0 F6 3460 BNE .2
B6A9- 4C BC A6 3470 .3 JMP $A6BC
B6AC- A2 FF 3480 .4 LDX #FFF
B6AE- 8E 5D B6 3490 STX APPEND.FLAG
B6B1- D0 F6 3500 BNE 3 ...ALWAYS
B6B3- 3510 .BS 29 <NOT USED>
3520 *-----
3530 * STRANGE CODE IN THE MIDDLE OF NOWHERE
3540 *-----
B6D0- 20 58 FC 3550 JSR MON.HOME CLEAR SCREEN
B6D3- A9 C2 3560 LDA #SC2 PRINT "B01-00"
B6D5- 20 ED FD 3570 JSR MON.COUT
B6D8- A9 01 3580 LDA #1
B6DA- 20 DA FD 3590 JSR MON.PREYTE
B6DD- A9 AD 3600 LDA #SAD
B6DF- 20 ED FD 3610 JSR MON.COUT
B6E2- A9 00 3620 LDA #0
B6E4- 20 DA FD 3630 JSR MON.PREYTE
B6E7- 60 3640 RTS
B6E8- 3650 .BS 21 <NOT USED>
3660 .OR $08FD
3670 .TA $0EFD
08FD- 00 36 3680 BT1.ADDR .DA $3600
08FF- 09 3690 BT1.N .DA #9
3700 *-----
3710 * SECOND STAGE OF BOOT
3720 *-----
3730 .OR $B700
3740 .TA $0F00
3750 BOOT.STAGE2
B700- 8E E9 B7 3760 STX IOB.SLOT16
B703- 8E F7 B7 3770 STX IOB.PRVSILT
B706- A9 01 3780 LDA #1
B708- 8D F8 B7 3790 STA IOB.PRVDV
B70B- 8D EA B7 3800 STA IOB.DRIVE
B70E- AD E0 B7 3810 LDA BT.N
B711- 8D E1 B7 3820 STA BT.CNT
B714- A9 02 3830 LDA #2
B716- 8D EC B7 3840 STA IOB.TRACK
B719- A9 04 3850 LDA #4
B71B- 8D ED B7 3860 STA IOB.SECTOR
B71E- AC E7 B7 3870 LDY BT.BT1+1
B721- 88 3880 DEY
B722- 8C F1 B7 3890 STY IOB.BUFFER+1
B725- A9 01 3900 LDA #1
B727- 8D F4 B7 3910 STA IOB.OPCODE
B72A- 8A 3920 TXA SLOT*16
B72B- 4A 3930 LSR GET SLOT #
B72C- 4A 3940 LSR
B72D- 4A 3950 LSR
B72E- 4A 3960 LSR
B72F- AA 3970 TAX
B730- A9 00 3980 LDA #0
B732- 9D F8 04 3990 STA $4F8,X
B735- 9D 78 04 4000 STA $478,X
B738- 20 93 B7 4010 JSR RW.PAGES
B73B- A2 FF 4020 LDX #FFF
B73D- 9A 4030 TXS EMPTY STACK
B73E- 8E EB B7 4040 STX IOB.VOLUME

```

```

B741- 4C 8B BF 4050 JMP $BFC8 PATCH TO SETVID AND CLOBBER
4060 * THE LANGUAGE CARD, IF IN SLOT 0
B744- 20 89 FE 4070 JSR MON.SETKBD
B747- 4C 84 9D 4080 JMP $9D84 DOS HARD ENTRY
4090 *
4100 * WRITE DOS IMAGE ON TRACKS 0-2
4110 *
4120 WRITE.DOS.IMAGE
B74A- AD E7 B7 4130 LDA BT.BT1+1 COMPUTE # OF PAGES
B74D- 38 4140 SEC
B74E- ED F1 B7 4150 SBC IOB.BUFFER+1
B751- 8D E1 B7 4160 STA BT.CNT
B754- AD E7 B7 4170 LDA BT.BT1+1 START AT END, WORK BACKWARD
B757- 8D F1 B7 4180 STA IOB.BUFFER+1
B75A- CE F1 B7 4190 DEC IOB.BUFFER+1
B75D- A9 02 4200 LDA #2 START ON TRACK 2
B75F- 8D EC B7 4210 STA IOB.TRACK
B762- A9 04 4220 LDA #4 SECTOR 4
B764- 8D ED B7 4230 STA IOB.SECTOR
B767- A9 02 4240 LDA #2
B769- 8D F4 B7 4250 STA IOB.OPCODE
B76C- 20 93 B7 4260 JSR RW.PAGES WRITE STAGE2 PART OF DOS
B76F- AD E7 B7 4270 LDA BT.BT1+1 SET UP BOOT SECTOR IMAGE
B772- 8D FE B6 4280 STA BT1.ADDR+1+$B600-$0800
B775- 18 4290 CLC COMPUTE STARTING ADDRESS OF WRITE
B776- 69 09 4300 ADC #9
B778- 8D F1 B7 4310 STA IOB.BUFFER+1
B77B- A9 0A 4320 LDA #10 WRITE 10 PAGES
B77D- 8D E1 B7 4330 STA BT.CNT
B780- 38 4340 SEC
B781- E9 01 4350 SBC #1
B783- 8D FF B6 4360 STA BT1.N+$B600-$0800
B786- 8D ED B7 4370 STA IOB.SECTOR
B789- 20 93 B7 4380 JSR RW.PAGES WRITE SECTORS 9-0 ON TRACK 0
B78C- 60 4390 RTS
4400 *
B78D- 00 00 00 4410 .HS 000000000000 <NOT USED>
B790- 00 00 00 4420 *
4430 * READ/WRITE A GROUP OF PAGES
4440 *
4450 * BT.CNT # OF SECTORS TO READ/WRITE
4460 * IOB SET UP FOR FIRST TS TO R/W
4470 *
4480 RW.PAGES
B793- AD E5 B7 4490 LDA BT.IOB+1 GET IOB ADDRESS
B796- AC E4 B7 4500 LDY BT.IOB
B799- 20 B5 B7 4510 JSR ENTER.RWTS READ/WRITE ONE SECTOR
B79C- AC ED B7 4520 LDY IOB.SECTOR IGNORE ERRORS IF ANY
B79F- 88 4530 DEY BACK UP SECTOR #
B7A0- 10 07 4540 BPL .1 STILL IN SAME TRACK
B7A2- A0 0F 4550 LDY #15 START WITH SECTOR 15 IN NEXT TRACK
B7A4- EA 4560 NOP
B7A5- EA 4570 NOP
B7A6- CE EC B7 4580 DEC IOB.TRACK BACKWARD THROUGH THE TRACKS
B7A9- 8C ED B7 4590 STY IOB.SECTOR
B7AC- CE F1 B7 4600 DEC IOB.BUFFER+1 DOWN ONE PAGE IN MEMORY
B7AF- CE E1 B7 4610 DEC BT.CNT ANY MORE PAGES TO DO?
B7B2- D0 DF 4620 BNE RW.PAGES YES
B7B4- 60 4630 RTS NO, RETURN
4640 *
4650 * ENTER RWTS
4660 *
4670 ENTER.RWTS
B7B5- 08 4680 PHP SAVE STATUS ON STACK
B7B6- 78 4690 SEI DISABLE INTERRUPTS
B7B7- 20 00 ED 4700 JSR RWTS CALL RWTS
B7BA- B0 03 4710 BCS .1 ERROR RETURN
B7BC- 28 4720 PLP RESTORE STATUS
B7BD- 18 4730 CLC SIGNAL NO RWTS ERROR
B7BE- 60 4740 RTS RETURN TO CALLER
B7BF- 28 4750 PLP RESTORE STATUS
B7C0- 38 4760 SEC SIGNAL RWTS ERROR
B7C1- 60 4770 RTS RETURN TO CALLER

```

```

4780 *-----
4790 *          SET UP RWTS TO WRITE DOS
4800 *-----
4810 SETUP.WRITE.DOS
B7C2- AD BC B5 4820 LDA FMP.SUBCOD IMAGE ADDRESS
B7C5- 8D F1 B7 4830 STA IOB.BUFFER+1
B7C8- A9 00 4840 LDA #0
B7CA- 8D F0 B7 4850 STA IOB.BUFFER
B7CD- AD F9 B5 4860 LDA FMW.VOLUME VOLUME #
B7D0- 49 FF 4870 EOR #FFF UNCOMPLEMENT IT
B7D2- 8D EB B7 4880 STA IOB.VOLUME
B7D5- 60 4890 RTS
4900 *-----
4910 *          CLEAR 256 BYTES STARTING AT ($42,43)
4920 *-----
4930 ZERO.CURRENT.BUFFER
B7D6- A9 00 4940 LDA #0
B7D8- A8 4950 TAY
B7D9- 91 42 4960 .1 STA ($42),Y
B7DB- C8 4970 INY
B7DC- D0 FB 4980 BNE .1
B7DE- 60 4990 RTS
5000 *-----
5010 *          PARAMETERS FOR SECOND STAGE OF BOOT PROCESS
5020 *-----
B7DF- 5030 .BS 1 <NOT USED>
B7E0- 1B 5040 BT.N .DA #27 # OF PAGES TO R/W (PARAMETER)
B7E1- 5050 BT.CNT .BS 1 # OF PAGES TO R/W (VARIABLE)
B7E2- 0A 5060 BT.IS .DA #10 1ST SECTOR # IN THIS STAGE
B7E3- 5070 .BS 1
B7E4- E8 B7 5080 BT.IOB .DA IOB ADDRESS OF IOB
B7E6- 00 B6 5090 BT.BT1 .DA BOOT.STAG1+$B600-$0800 ADDR OF 1ST STAGE BOOT
5100 *-----
5110 *          IOB FOR RWTS CALLS
5120 *-----
5130 IOB
B7E8- 5140 IOB.TYPE .BS 1 0-MUST BE $01
B7E9- 5150 IOB.SLOT16 .BS 1 1-SLOT # TIMES 16
B7EA- 5160 IOB.DRIVE .BS 1 2-DRIVE # (1 OR 2)
B7EB- 5170 IOB.VOLUME .BS 1 3-DESIRED VOL # (0 MATCHES ANY)
B7EC- 5180 IOB.TRACK .BS 1 4-TRACK # (0 TO 34)
B7ED- 5190 IOB.SECTOR .BS 1 5-SECTOR # (0 TO 15)
B7EE- FB B7 5200 IOB.PNIDCT .DA DCT 6-ADDRESS OF DCT
B7F0- 5210 IOB.BUFFER .BS 2 8-ADDRESS OF DATA
B7F2- 5220 IOB.SECTSZ .BS 2 10-# BYTES IN A SECTOR
B7F4- 5230 IOB.OPCODE .BS 1 12-0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
B7F5- 5240 IOB.ERROR .BS 1 13-ERROR CODE: 0, 8, 10, 20, 40, 80
B7F6- 5250 IOB.ACTVOL .BS 1 14-ACTUAL VOLUME # FOUND
B7F7- 5260 IOB.PRVSLT .BS 1 15-PREVIOUS SLOT #
B7F8- 5270 IOB.PRVDIV .BS 1 16-PREVIOUS DRIVE #
B7F9- 5280 .BS 2
B7FB- 00 01 EF 5290 DCT .HS 0001EFD8
B7FE- D8 5300 .BS 1
B7FF-

```

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$12 per year in the U.S.A., Canada, and Mexico. Other countries add \$12/year for extra postage. Back issues are available for \$1.20 each (other countries add \$1 per back issue for postage). All material herein is copyrighted by S-C SOFTWARE, all rights reserved. Unless otherwise indicated, all material herein is authored by Bob Sander-Cederlof. (Apple is a registered trademark of Apple Computer, Inc.)